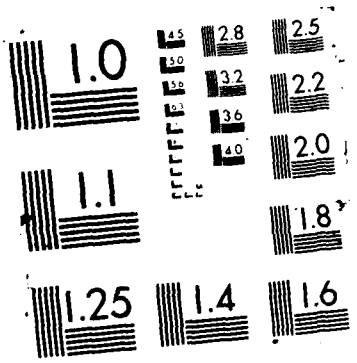
AD-A186 411 CRITERIA FOR IDENTIFYING SOFTWARE ENGINEERING ENVIRONMENT RELEASES SEE-RE..(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA T E LINDQUIST JUL 85 IDA-P-1856 1/1

UNCLASSIFIED IDA/HQ-85-30182 MDA903-84-C-0031 F/G 12/8 NL


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8

IDA PAPER P-1856

CRITERIA FOR IDENTIFYING SOFTWARE ENGINEERING ENVIRONMENT RELEASES SEE-RELID-002

Timothy E. Lindquist

July 1985

Prepared for

Office of the Under Secretary of Defense for Research and Engineering

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, VA 22311

IDA Log No. HQ 85-30182

SECURITY CLASSIFICATION OF THIS PAGE

AD-A186 411

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY [redacted]	3. DISTRIBUTION/AVAILABILITY OF REPORT Public release/unlimited distribution.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE n/a	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) IDA Paper P-1856	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION OUSDRE (DoD IDA Management Office)
6c. ADDRESS (City, State, and ZIP Code) 1801 N. Beauregard St. Alexandria, VA 22311	7b. ADDRESS (City, State, and ZIP Code) 1801 N. Beauregard St. Alexandria, VA 22311
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of Under Secretary of Def., R&E 8b. OFFICE SYMBOL (If applicable) OUSDRE(R&AT)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84 C 0031
8c. ADDRESS (City, State, and ZIP Code) STARS Joint Program Office 1211 Fern St. Arlington, VA 2q202	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. T-5-284 / WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification)

Criteria for Identifying Software Engineering Environment Releases

12. PERSONAL AUTHOR(S).
Timothy Lindquist

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM _____ TO _____	July 1985	13

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Software Engineering Environments; Computers, Computer Programming; Software Releases; Software Tools; Standardization

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper is one of a series on the development of a DoD Software Engineering Environment (SEE). It illuminates the design and implementation issues that underlie determining what components should be included in successive releases of a SEE. The paper presents a set of studies that identify the needed information, and discusses how the information may be combined to define releases. The studies include gathering information relevant to the environment's potential customers, development requirements, and interfacing needs. This information is combined into a list of components with attributes that can then be used to identify specific release elements.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
■ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

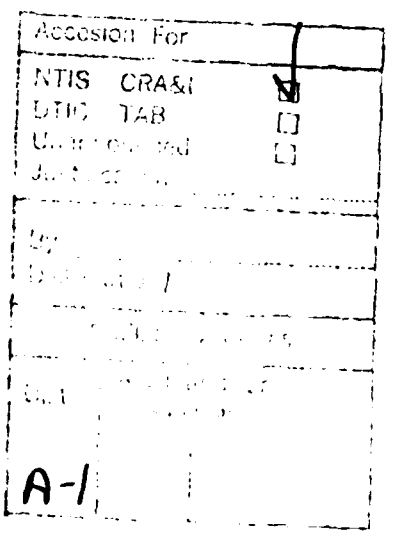
SECURITY CLASSIFICATION OF THIS PAGE

IDA PAPER P-1856

CRITERIA FOR IDENTIFYING SOFTWARE ENGINEERING ENVIRONMENT RELEASES SEE-RELID-002

Timothy E. Lindquist

July 1985



INSTITUTE FOR DEFENSE ANALYSES

Contract No. MDA 903 84 C 0031
Task T-5-284

TABLE OF CONTENTS

1.0 INTRODUCTION

When planning releases for a Software Engineering Environment (SEE), several issues, most of which interact, must be considered. To assure that all relevant issues are addressed and resolved in the best manner possible, a structured approach to defining releases should be adopted. In this paper we address, for a SEE, the problem of matching design and implementation issues with the pragmatics of greatest need, development cost, and availability of environment components. A SEE is a comprehensive set of tools and methodologies supporting all activities associated with the life-cycle of software. As with any large complex software system for which new technology continually develops, a SEE must be developed and adopted in stages. As technology advances the state of the art of software development and support, the environment must accommodate changes and extensions. For economic, training, and extensibility reasons, software engineering environments are typically constructed and distributed as a sequence of releases.

The purpose of this paper is to illuminate the issues that underlie determining what components should be included in successive releases of a SEE. The paper presents a set of studies that identify the needed information, and discusses how the information may be combined to define releases. The studies include gathering information relevant to the environment's potential customers, development requirements, and interfacing needs. This information is combined into a list of components with attributes that can then be used to identify specific release elements.

Ultimately, any environment is concerned with improving the software process, improving software productivity, and improving software quality. Releases of a SEE must directly address these needs, while still providing a usable environment in the short term. To do so requires a proper mix in each release of introducing new technology and planning for future maturation and extension. To accomplish the maturation and extension goals, early releases of a SEE must define appropriate interfaces. Adding environment components can be accomplished with a higher degree of integration if interfaces such as the CAIS (Common Ada Interface Set) exist. Replacement or improvement of components can also be accomplished more readily through the use of predefined interfaces. For example, a compiler's use of DIANA (Descriptive Intermediate Attributed Notation for Ada) as an intermediate form would facilitate the replacement of its back-end. Accordingly, a SEE release has two aspects. The first aspect is the set of components and methodologies constituting the environment together with associated support materials (such as documentation). The components are the actual software elements, the methodologies and support materials indicate how the environment is to be used. The second aspect of a release is a set of tool building tools, interfaces, or associated guidelines for environment

extension and change. These tools allow another vendor, for example, to further develop the environment in a consistent manner.

Three different categories of issues relating to environment releases are identified and discussed in this paper. These categories include structural, management, and user issues. Structural concerns represent the needs of the tool builder and the environment architect, and management concerns are those of the environment owner. Finally, user concerns are those centered on the different roles of environment use (the user's view). This paper is organized according to these categories. In three successive sections each of these sets of issues is discussed. Following these sections is a presentation of the process to be used to select releases. The process is based on conducting three studies. Releases are identified by balancing the factors which result from the studies.

2.0 STRUCTURAL CONCERNS

Under the heading of Structural Concerns we discuss issues which relate to the Tool Writer's and Architect's view of a SEE. These two sets of issues form the basis for addressing the implementation complexity of the environment. Although implementation issues are related to those discussed below, such as component availability, they are treated separately because the resolution of these issues provides direct input to the environment's implementor. The tool writer is taken to be anyone who's role is to add or modify functional components of a release. Thus, issues dealing with implementing various environment components are called tool writer concerns. These concerns may include the suitability of the implementation language to the problem of coding an environment or the time complexity of the implementation of a release component. The environment architect's view is from a level higher than that of the tool writer's. The architect is concerned with the relationships among tools and the manner with which tools fit together. For example, the architect may be concerned with not excluding possible extensions, and consequently might impose certain constraints on releases.

Tool Writer. If a release were defined strictly according to the needs of implementation then relevant constraints such as marketability and transition from release to release would not receive adequate attention. Alternatively, attention must be paid to the implementation when defining releases. At the worst, a release defined and implemented without resolving tool writer concerns will be excessive in cost and implementation complexity. Two tool writer concerns are discussed in this section including:

1. Functional implementation hierarchy (USES relation)

2. Component coupling (uses the results of relation)

Functional Implementation hierarchy. The functional hierarchy is an ordering of implementation components. The ordering is determined by one component's use of another. Several interactions exist among components of an environment, and one form is the functional hierarchy formed by the component calling structure.

A software engineering environment can be classified from several different views. One approach is to construct a taxonomy based on the functionality viewable to the user of the environment. This eliminates a majority of the kernel level facilities that are included as computational functions. The view used in the Functional Implementation Hierarchy is to construct a taxonomy of environment components based on the functionality performed by the software constituting the environment. Each computation module, whether user viewable or not, is represented. A dependency structure is defined based

upon each component's use of others (the USES relation). All computational functions performed by the environment are listed in the taxonomy. As a third and mixed approach, an environment can be defined by various levels, which may correspond to an implementation technique or to an abstract model of functionality (such as the ISO model for network interconnections).

One tool builder constraint can be seen through the Functional Implementation Hierarchy. The hierarchy is defined to include, the functional components of the environment to the level of kernel interfaces. Thus, modules supporting

1. File manipulation,

2. Process management,

3. Source program intermediate form manipulation, and

4. User viewable functions, such as configuration control,

are all examples of facilities that would be included in such a hierarchy. A constraint based on this hierarchy is that each release be complete with respect to the USES hierarchy, and that a release maximize the number of user viewable components as compared to underlying components. Thus, the user viewable components of a release should have the maximum number of common components. This constraint supports the approach to building systems in which a set of primitives are constructed as the foundation of the system. User viewable facilities are then constructed through the use of the primitives. The tool writer would define a release in which a relatively small set of primitives would be optimally used by tools and tool features. As an example of this constraint, a release would not be defined to include a screen based editor without employing the primitives needed for screen management in other tools (such as help facilities and command recognition).

Interactions. Another tool builder concern deals with the coupling among environment components. This requirement identifies the complexity of interactions and protocols between tools of the environment, which is called the degree of coupling between tools. Intertool coupling refers to the data communications that exist between two or more tools. For example, compilers and linkers on typical systems are coupled together by virtue of the data protocol used to communicate the object module from the compiler to the linker. Although these tools may use the same set of underlying facilities to create and use an object module, the coupling exists distinct of the facilities. Information required by one is produced, in a specific format, by the other.

Besides tool-to-tool coupling, other types of interactions exist within an environment's components. One such example is

the dependence that a kernel level facility such as that to invoke a new process has on the linker. Although Invoke is not a user viewable function, the image format produced by the linker must be recognizable by the Invoke facility, and the image must contain all the information needed to cause the program to execute.

The release constraint imposed by interactions is that releases must be complete with respect to satisfied protocols. Releases must also be defined in such a way that interactions are minimized.

Architect's View. The architect of a software engineering environment is concerned with the composite of tools included in a release, with growth of the environment from one release to another, and with the interfaces defined to the environment.

Various approaches are valid for the growth of an environment from release to release. One example is an evolution path from a prototype of the environment via successive refinements to more complete capability. Another option approximates the big-bang approach. Here, a small number of releases are defined each making large incremental jumps toward more complete capability. One final option mentioned is one of successive adaptations. Starting with an existing baseline environment, changes and enhancements are made in series of releases to arrive at a final full capability release.

For each of the growth scenarios mentioned above, a concern from one release to another is the degree of extensibility built into releases. To a large extent, the front-end work to build in consistency, commonality, and extensibility can greatly reduce the effort required to move from one release to the next.

To provide for these goals, information interfaces must be defined among tools. Examples of such interfaces might be:

1. A standard interface to database facilities,
2. A standard associated with the user's interface,
3. A standard set of kernel facilities to support tools (CAIS),
4. A standard interface for intermediate code, and
5. A standard default text file format for tool communication.

The goal of defining such standard interfaces is to encourage continued tool development and enhancement in a consistent manner and to allow successive releases to redefine components more gracefully. Additionally, such interfaces encourage more integrated tool sets and increased commonality among application specific or special purpose tools.

3.0 MANAGEMENT CONCERNS

Management issues are considered to be concerns from the view of the environment owner. The owner is interested in the cost of developing, maintaining, and using the SEE. Further, the long term impact on software costs, software quality, and productivity must all be considered. Questions addressing the marketability of each environment release must also be addressed. The greatest potential for payoff, willingness to adopt an environment, and areas of greatest need are all marketing considerations that must be addressed in defining releases.

Defining a set of releases for which there is a clear migration path for the highest potential users is of critical importance. Tools to support conversion from an existing system are needed. Direct interfaces to existing tools and interoperability with data manipulated by existing tools is also necessary. From the standpoint of acceptance, early releases must contain substantial support for tutorials and help facilities. Tools that allow interfaces with software written in various languages provide more ready acceptance and easier transition for projects under development.

Another management issue to be considered is component availability. This requirement addresses the level of risk involved in individual components of the release and to a lesser extent the cost of components. The requirement minimizes risks relating to state-of-the-art technology and maximizes the use of existing software and interfaces. Levels of availability may be defined to include the following:

1. Technology does not exist,

2. Technology exists, but is not yet well developed,

3. Existing components may be adapted, and

4. Existing components may be directly used.

The economic requirement is that each release shall maximize the level of availability of its components. Another requirement, addressing the insertion of technology, indicates that a predetermined level of maturity of technology be present in a release.

4.0 USER CONCERNS

While the previous sections have identified criteria for releases that are related directly to the implementation of the release, another set of concerns deal with defining releases from the perspective of the environment's user. Isolating from concern the technical aspects of an environment, releases may be defined solely with respect to user viewable functionality. Users of a SEE are concerned with it helping them by increasing profit, producing a better product, and satisfying needs as quickly as possible. The user's interests are to have the environment used by as much of the user population as possible, and to increase user's productivity.

Releases must be defined using role based use-sequences as the focal for identifying needed functionality. For each user-role, an analysis of the computing functions needed to perform role tasks is conducted. Based on role analysis, functionality sufficient to accomplish certain tasks is identified. Use sequences at as low a level as FINDLINE before CHANGELINE and WRITEFILE before EXITEDITOR, for example, could be used to identify functionality. While this requirement might identify an overly ambitious initial release, it does provide for completeness of the user's interface(s) for a single role. In addition to completeness, consistency among command formats, device usage, and command options is another example of the type of user concerns which must be addressed in formulating releases.

5.0 RESOLVING THE CONCERNS THROUGH STUDIES

Release decisions can be made for those criteria which are not in conflict, however, further studies aimed at more completely elaborating tradeoffs would greatly increase the effectiveness of releases. Three studies are recommended, including a component availability study, a market analysis, and an interface standards study.

An availability study is aimed at identifying and gathering information on software components relevant to the SEE. This study collects information on the variety of components available and examines the current state-of-the-art technology of those components. Included in the study is a classification of each functional component on the availability scale listed above (ranging from Technology Does Not Exist to Existing Components May Be Directly Used). Additionally, the study includes, for each user viewable component, an analysis of the high-risk to low-risk range representing advances in design/implementation of the component. Input to the study consists of documentation on existing environments, case studies regarding their use, information on developing environments, and materials describing experimentation in environment technology. The study constructs listings and classifications of environments and their components indicating the availability and risk of each. The result of the study may report in combinations such as components, features of components, and methodologies.

Another survey identifies a marketing strategy for an environment. The strategy considers issues such as whether releases should be based on a methodology or user roles and whether releases should be depth first or breadth first. That is, a given release may identify a single role or methodology component and completely address that component; this is depth first. Alternatively, a single release may address all roles to some consistent level of treatment; this is breadth first. The marketing study addresses the desired differences between releases and the frequency with which releases occur. The relative level of risk of components within a release is also addressed, which identifies the extent that new technology is included in each release. The survey also identifies those areas containing greatest potential for productivity improvement as those most important to be included in early releases. The market analysis generates a list of highest potential users of releases. For each user, a list of their software development problems is developed. Using the results of the availability study, an estimation of the improvement to a user's software process is constructed. For a sample of potential users, the study produces a market index of each element from the availability study.

The interface study examines the questions: what level of uniformity should exist among interfaces and to what extent

should standard interfaces be used in an environment? This study is aimed at deciding what components of an environment should be directly built by the owner and to what extent the computing industry can be encouraged to build components. Further, the study identifies the proper mechanisms to provide consistency and uniformity among industry supplied and owner supplied components. This study also identifies which interfaces must be standardized in releases. As mentioned, potentials include a graphics interface, a database interface, an interface to kernel facilities (CAIS), an intermediate program format, and standards for the user interface.

The results of these studies are used with structural concerns and estimates of implementation complexity to identify releases. An environment may define any number of releases depending on the intended increment from one release to the next. The increment may vary from small enhancements between releases to major changes from one release to the next. An initial release is defined from the availability study and market analysis. This release is then tuned according to structural concerns. The initial release must reflect a decision as to the extent of life-cycle activities that should have automated support.

6.0 REFERENCES

Draft Military Standard: Common APSE Interface Set (CAIS), Project IPSC/ECRS 0208, January 1985.

Joint Service Software Engineering Environment (JSSEE) Operational Concept Document (OCD), JSSEE-ecd-00.2, November 1984.

Kean, E.S. and Lamonica, F.S., A Taxonomy of Tool Features for a Life-cycle Software Engineering Enviornment, Report RADC/COEE Griffiss AFB, NY, SEE-RELID-001, April 1985.

Nash, S.H. and Redwine, S.T., Information Interface Related Standards, Guidelines, and Recommended Practices SEE-INFO-004, Institute for Defense Analyses, Paper P-1842, July 1985.

Distribution List for IDA Paper P-1856

DoD

Col. Joe Greene — 10 copies
Director, STARS Joint Program Office
1211 Fern St., C-107
Arlington, VA 22202

Other

Dr. Tim Lindquist — 2 copies
Associate Professor
Computer Science Department
Arizona State University
Tempe, AZ 85287

Defense Technical Information Center — 2 copies
Cameron Station
Alexandria, VA 22304-6145

CSED Review Panel

Dr. Dan Alpert, Director — 1 copy
Center for Advanced Study
University of Illinois
912 W. Illinois Street
Urbana, Illinois 61801

Dr. Barry W. Boehm — 1 copy
TRW Defense Systems Group
MS 2-2304
One Space Park
Redondo Beach, CA 90278

Dr. Ruth Davis — 1 copy
The Pymatuning Group, Inc.
2000 N. 15th Street, Suite 707
Arlington, VA 22201

Dr. Larry E. Druffel — 1 copy
Software Engineering Institute
Shadyside Place
480 South Aiken Av.
Pittsburgh, PA 15231

Dr. C.E. Hutchinson, Dean — 1 copy
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755

Mr. A.J. Jordano — 1 copy
Manager, Systems & Software
Engineering Headquarters
Federal Systems Division
6600 Rockledge Dr.
Bethesda, MD 20817

Mr. Robert K. Lehto — 1 copy
Mainstay
302 Mill St.
Occoquan, VA 22125

Mr. Oliver Selfridge — 1 copy
45 Percy Road
Lexington, MA 02173

IDA

Gen. W.Y. Smith, HQ
Mr. Seymour Deitchman, HQ
Ms. Karen Webber, HQ
Dr. Jack Kramer, CSED
Dr. John Salasin, CSED
Dr. Robert Winner, CSED
Ms. Katydean Price, CSED — 2 copies
IDA C&D Vault — 3 copies

END

12-87

DTIC